Exploring weak scalability for FEM calculations on a GPU-enhanced cluster

Dominik Göddeke [a,*,1], Robert Strzodka [b,2],
Jamaludin Mohd-Yusof [c], Patrick McCormick [c,3],
Sven H.M. Buijssen [a], Matthias Grajewski [a] and Stefan Turek [a]

[a] *Institute of Applied Mathematics, University of Dortmund*

[b] *Stanford University, Max Planck Center*

[c] *Computer, Computational and Statistical Sciences Division, Los Alamos National Laboratory*

Abstract

The first part of this paper surveys co-processor approaches for commodity based clusters in general, not only with respect to raw performance, but also in view of their system integration and power consumption. We then extend previous work on a small GPU cluster by exploring the heterogeneous hardware approach for a large-scale system with up to 160 nodes. Starting with a conventional commodity based cluster we leverage the high bandwidth of graphics processing units (GPUs) to increase the overall system bandwidth that is the decisive performance factor in this scenario. Thus, even the addition of low-end, out of date GPUs leads to improvements in both performance- and power-related metrics.

Key words: graphics processors, heterogeneous computing, parallel multigrid solvers, commodity based clusters, Finite Elements
PACS: 02.70.-c (Computational Techniques (Mathematics)), 02.70.Dc (Finite Element Analysis), 07.05.Bx (Computer Hardware and Languages), 89.20.Ff (Computer Science and Technology)

* Corresponding author. Address: Vogelpothsweg 87, 44227 Dortmund, Germany. Email: dominik.goeddeke@math.uni-dortmund.de, phone: (+49) 231 755-7218, fax: -5933

[1] Supported by the German Science Foundation (DFG), project TU102/22-1

[2] Supported by a Max Planck Center for Visual Computing and Communication fellowship

[3] Partially supported by the U.S. Department of Energy's Office of Advanced Scientific Computing Research.

1 Introduction

Commodity based clusters dominate the Top500 supercomputer list in the number of deployed systems [21]. The mass production of commodity components offers advantages in acquisition costs, availability and extensibility, modularity and compatibility. However, these metrics do not tell the whole story. The inefficiency of today's CPUs for high performance computing (HPC) tasks in general and Finite Element computations in particular leads to low performance and high power consumption per node, requiring many nodes for large problems and thus additional infrastructure for accommodating, connecting and cooling that many components. In addition, very high numbers of nodes cause the administration and maintenance costs to increase super-linearly.

Several research projects and companies have attempted to increase the performance and reduce the relative power consumption of commodity based clusters with more specialized co-processors. As the application area of these co-processors is restricted, the hardware designers have more freedom in the exploration of the different tradeoffs in designs:

- General purpose HPC vs. application-specific hardware.

  The former caters to the needs of more customers, promising larger production lots; the latter allows higher performance and power efficiency for a specific application. In this paper we consider the general purpose approach. However, general purpose here refers to various HPC workloads , not the full diversity of all CPU applications.
- Low vs. high memory bandwidth to the co-processor.

  Data processing capabilities usually exceed data transfer capabilities. Improving compute performance (more transistors) is much cheaper than improving memory performance (pin limits). Data intensive applications such as sparse linear algebra problems benefit greatly from high bandwidth access to the co-processor, but such connections inflate the power budget. In particular, power-aware co-processors must restrict the off-chip bandwidth and thus achieve high efficiency only on compute intensive applications like dense linear algebra problems. As we deal with sparse linear algebra problems in this paper, a high bandwidth connection to the co-processor is required.
- Processor vs. system interconnects for the co-processors.

  Examples for the former are Hypertransport couplings of CPUs and FPGAs. In such couplings the co-processor has little or no dedicated off-chip memory and shares the main memory with the CPU, see Figure 1 on the left. Depending on the co-processor, the connection sometimes allows direct interaction with the CPU's cache. The integration can go even further, forming a System-on-Chip (SoC) on the same die. In this paper, we are interested in co-processors that can be connected as add-on boards via standardized slots, typically PCI/PCI-X/PCIe. When working with co-processors with high bandwidth to their on-board memory (see previous item), the bus to the host system becomes a bottleneck as shown in Figure 1 on the right. Strategies for dealing with this bottleneck are discussed in Section 2.3.

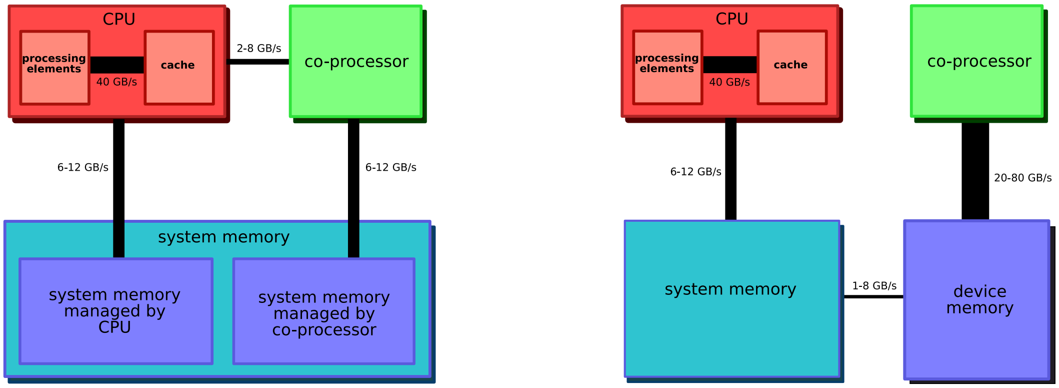

Fig. 1. Interconnects and bandwidth between host and coprocessor. Left: processor interconnects. Right: System interconnects.

Adding co-processors to an existing homogeneous commodity based cluster turns it into a heterogeneous system. In this context we distinguish between *local* and *global* heterogeneity. In the case of solely local heterogeneity the nodes contain different types of processors but the hardware configuration of all nodes is the same. Global heterogeneity means that the nodes differ in their configurations, e.g. some have been enhanced with an accelerator of some kind and others with a different one or not at all. As we concentrate on the scalability of a GPU enhanced cluster in this paper, we assume that all nodes are configured equally.

In this case the co-processor code can be tested on a single node and later the same code can be run on each node in the cluster. We apply a *minimally invasive* integration of the accelerator into an existing MPI-based code: The changes are reduced to a minimum and the hardware dependent code is accessed through the same calls as the CPU code. For the user it is sufficient to change a parameter file to enable hardware acceleration and no modifications to application code are required. More details on the hardware integration and remaining limitations of our cluster are discussed in Section 2.3 and 2.4.

1.1 Paper overview

This introductory section presents related work on co-processors for HPC clusters. The necessary background on graphics processors and challenges in building GPU clusters are discussed in Section 1.3. Section 2.1 lists the hardware details of the cluster we use for our scalability tests. Sections 2.2 and 2.3 describe our software approach for including GPU acceleration in a minimally invasive way . We address limitations of the available GPUs in Section 2.4. Section 3 contains the discussion of our results with respect to weak scalability, bandwidth, acceleration, power consumption and portability of our approach. We conclude with suggestions for future work in Section 4 and a comprehensive bibliography.

1.2 Heterogeneous clusters and related work

In this section we present related work on heterogeneous HPC clusters that have been successfully deployed. We discuss them in three classes depending on the specialization area of the co-processors.

- Clusters with dedicated HPC accelerators.

  These clusters contain co-processors targeted explicitly at HPC workloads. This class incurs higher acquisition costs due to smaller volume but as the entire product is focusing on HPC deployment the software integration of these hardware accelerators is smoother. Moreover, a strong emphasis is put on power efficiency and reliability, which is particularly important for large-scale systems and offers the possibility to significantly improve the performance/watt ratio of clusters. Two prominent representatives of this class are:

  · GRAPE supercomputer, MDGrape-3 supercomputer.

  The GRAPE series of processors is highly specialized for certain computation patterns arising in the solution of astrophysical (GRAPE) and molecular dynamics (MDGrape) problems. Application-specific hardware like this greatly outperforms general purpose hardware in both performance and power efficiency. The newest generation, the MDGrape-3 deployed at Japan's Research Institute of Physical and Chemical Research (RIKEN) in 2006, is the first system to perform more than one petaflops. Because of its specialization, the MDGrape-3 chip consumes only 0.1 Watt per gigaflop, but the price for this efficiency is the restriction to the designated computation patterns [11].

  · TSUBAME cluster accelerated by ClearSpeed Advance Accelerator Boards.

  The TSUBAME cluster at the Tokyo Institute of Technology was upgraded with these boards in 2006 [34]. With the help of a highly tuned implementation of the BLAS library tailored for the co-processors, the cluster now delivers a 24% increase in LINPACK performance while requiring only 1% more overall power [6]. Consequently, no upgrade to the cooling system was necessary and additional maintenance is kept to a minimum. The two CSX600 chips on the boards (which are connected via PCI-X) are general purpose and very power efficient; together with 1 GiB of memory connected at 6.4 GB/s they consume only 25 Watt while delivering 50 gigaflops in double precision. Due to the connection bandwidth, the chips perform best at compute intensive applications and are less suitable if data movement dominates the problem.

- Clusters with FPGAs.

  Field Programmable Gate Arrays (FPGAs) can be configured to perform arbitrary computations in hardware, which offers many possibilities for application acceleration ranging from low level hardware pipeline design for certain time critical computations to high level application specific instruction set extensions of a soft processor core running in the FPGA. The main strength of FPGAs is the very high spatial parallelism and connectivity; performance gains and power efficiency thus depend on the exploitation of these features. This poses a software development problem as the spatial/structural approaches to FPGAs conflict with the temporal programming models for CPUs. Application programmers are usually not willing to resolve such problems themselves and thus commercially available systems either target certain application areas and provide hardware accelerated libraries for them or greatly simplify the design flow and deduce the spatial configuration from temporal high level languages. One example of this approach is the Cray XD1 Supercomputer [7].

- Clusters with multimedia co-processors.

  These co-processors originate from the large graphics, video and gaming market that generates very favorable price/performance ratios. Although high bandwidth connections to the co-processors dissipate a lot of power, the resulting high performance on data intensive applications can still improve the power efficiency of a cluster (cf. Section 3.3). While multimedia and HPC applications share common requirements, user interfaces can differ significantly and a transition layer must be designed to reformulate the problem in HPC terms. Three examples of these types of clusters are:

  · Cell BE cluster.

    The Cell BE processor [29] ships either in blades (offered for instance by IBM), in PCIe add-on cards (offered by Mercury Computer Systems), or in the PlayStation 3 gaming console from Sony (see next paragraph). Mercury's Cell Accelerator Board delivers 179 gigaflops in single precision and consumes up to 210 Watts, including 1 GiB[4] of high-bandwidth XDR memory, 4 GiB of memory connected with a 22.4 GB/s bus to the Cell BE chip and its own Gigabit ethernet interface [20]. Prototype clusters with Cell processors have been demonstrated since 2005.

  · PlayStation cluster.

    The National Center for Supercomputing Applications and the computer science department at the University of Illinois connected 65 PlayStation 2 consoles in 2003 to form a Linux HPC cluster [23]. First prototypes of Linux clusters comprising PlayStation 3 consoles have been assembled [4, 5, 22]. In addition, Terra Soft Solutions offers PlayStation clusters with

[4] International standard IEC60027-2: G= 10^9, Gi= 2^{30} and similar for Ki, Mi.

a complete developer toolchain commercially.

- GPU cluster.

  GPU-enhanced systems have traditionally been deployed for scientific visualization. One of the largest examples is ’gauss’, a 256 node cluster installed by GraphStream at Lawrence Livermore National Laboratory [15]. Several parallel non-graphics applications have been ported to GPU clusters: Fan et al. present a parallel GPU implementation of flow simulation using the Lattice Boltzmann model [10]. Their implementation is 4.6 times faster than an SSE-optimized CPU implementation, but they do not discuss the impact of GPUs’ limited precision. The parallel efficiency degrades to 66% on 32 nodes. Stanford’s Folding@Home distributed computing project has recently deployed a dual-GPU 16 node cluster achieving speed-ups of 40 over extremely tuned SSE kernels [26].

After the successful application of the above co-processors in mid-range clusters with a few dozen nodes, these hardware configurations can be considered for large-scale systems. However, apart from the TSUBAME cluster, this has not yet been attempted, because of scalability and return-on-investment concerns. To the best of our knowledge, the ’gauss’ system (the only GPU cluster as large as ours) is solely used for visualization. In the TSUBAME cluster the ClearSpeed co-processor driver intercepts the standard calls of BLAS or FFT libraries and replaces them with hardware accelerated versions. Modifications of the application code are not required and the return-on-investment can be estimated in advance by counting library calls. The domain of dense linear algebra is very well suited for this approach.

For sparse matrices, which typically arise in Finite Element simulations, there is less standardization and the integration of hardware co-processor becomes more challenging. The reluctance to deal with the non-standard hardware-software interaction is usually so high that even promises of significant speed-ups in heterogeneous systems have not been convincing enough to deploy large-scale systems. In Section 2.2 we will outline our strategy for parallel multigrid solvers with decoupled local smoothing, which allows the integration of hardware accelerators without changes to application codes, and minimal modifications otherwise.

1.3 Graphics processor units and graphics clusters

Instead of another introduction to general purpose computations on graphics processors (GPGPU), we address several common misconceptions. For readers new to the subject, many elaborate introductory articles are available [16, 25, 31]. The community website has a wealth of information, including a paper archive, conference tutorials and code examples [14].

Allegedly, the limitation of GPUs to single precision has prevented broad adoption in scientific computing. While this is certainly true for some applications, the high number of lower precision processing elements can be exploited even in view of high accuracy requirements. Mixed precision methods and high precision emulation techniques have both been demonstrated to match the co-processor approach very well [8, 13, 33]. Moreover, both AMD and NVIDIA have already annouced double precision lines of their products.

GPUs also have a reputation of being hard to program for non-graphics applications. First, the algorithms have to be adapted to the parallel programming paradigm, but this is necessary in general for parallel architectures, e.g. multicore processors. Second, early adopters of GPGPU had to struggle with the coding of scientific applications in term of graphics primitives. Over the past few years, several academic and commercial programming abstractions and interfaces have emerged, greatly simplifying programming of GPUs:

- The vendors themselves provide either a low-level interface to the hardware, allowing direct access to the processing elements in an assembly-like fashion (CTM by AMD/ATI [27]), or a C-like programming environment that basically allows the program to spawn threads with limited inter-thread synchronization and communication capabilities across the processing elements (CUDA by NVIDIA [24]).
- BrookGPU, Sh and Accelerator are three examples of compilers and libraries for stream programming that hide most of the graphics-specific details and let the programmer focus on developing parallel algorithms [3, 19, 32].
- Similarly, several companies like Acceleware and RapidMind have started to offer programming environments and combined hardware software solutions to leverage GPUs.

Building a GPU cluster from scratch based on the workstation form factor (3U, rack-mountable) using commodity components is slightly more complicated than building a conventional cluster. When designing the nodes, one has to consider the GPUs in the power budget and infrastructure, as current high-end models consume as much as 140 W (G80, NVIDIA) or 180 W (R600, ATI), with high amperage loads, and many clusters are designed to run two or more boards per node. Money is well invested on reliable cooling solutions and the air flow design inside the nodes is even more important than for conventional clusters. Careful planning is necessary when deciding on the mainboards and the graphics boards: To increase the density of the cluster installation, preferably more than one GPU needs to be installed in each node, requiring at least two PCIe slots with full x16 support to avoid creating an artificial bottleneck. Professional graphics boards (FireGL/FireStream models by ATI, Quadro series by NVIDIA) allegedly promise higher MTBF and MTBM, but cost more than twice as much as equivalent consumer cards. Many vendors offer more dense GPU clusters optimized for visualization purposes.

Special cases are NVIDIA's QuadroPlex visual computing system and the Tesla GPU computing solution, both of which are high density solutions comprising up to four high-end GPUs in an external chassis with its individual power supply, connected to the host node via one PCIe x16 card and a cable. The server model of the QuadroPlex is only 1U high, consumes 1.2 kW and contains four Quadro FX 5600 GPUs with 1.5 GiB of memory each. The designs include proprietary NVIDIA chips that multiplex the PCIe lanes and handle the routing of traffic to the GPUs. Thus, full bandwidth to each of the GPUs is available as long as data transfers are not performed simultaneously. These designs are especially useful when upgrading an existing cluster with GPUs, as pure CPU-based clusters are often very dense installations, and adding GPUs to 1U nodes after the fact is largely impossible.

Finally, the technical numbers of GPUs alone often generate unrealistic expectations about the capabilities of the hardware. For instance, AMD's R600 chip can perform multiply-add instructions at 480 gigaflops and offers 74 GB/s of bandwidth for streaming access, the cached performance is even better [30]. While synthetic benchmarks indeed come close to these numbers, actual applications are not a chain of these ideal configurations, similar to CPUs. Moreover, note that the ratio of the peak numbers gives 480 gigaflops/74 GB/s = 26 flop/4B, which means that the peak streaming bandwidth and the peak computation are balanced if 26 operations are performed for each read/write access to an off-chip single float (4B). Data intensive applications such as the Finite Element computations we target require far fewer operations on each data item. For instance, our matrix-vector multiplication (cf. Section 2.2) has an *arithmetic intensity* of one. Therefore the speed-up on the GPU depends on the superiority of the graphics cards' *bandwidth* instead of the *parallelism* in the GPUs. This is not to imply that it is impossible to achieve significant performance improvements with GPUs but rather that a factor of 5 is much more common than 20 when comparing with an optimized CPU program [25].

2 System Components and Solution Scheme

2.1 Hardware configuration

For the test cases in this paper we use 160 compute nodes and one master node of the DQ visualization cluster at Los Alamos National Laboratory's Advanced Computing Lab. Each node has the following configuration:

CPU: Dual Intel EM64T, 3.4 GHz, 1 MiB L2 cache, 800 MHz FSB, 600 W power supply,

RAM: 8 GiB (5.5 GiB available), 6.4 GB/s shared bandwidth (theoretical),

INTERCONNECT: PCI-X based InfiniBand cards with 1.25 GB/s peak (∼780 MB/s benchmarked);
GPU: NVIDIA Quadro FX1400, 350 MHz, max. 75 W,
RAM: 128 MiB, 19.2 GB/s bandwidth,
INTERCONNECT: PCIe bus with 4 GB/s peak (∼1.2 GB/s benchmarked).

The InfiniBand interconnect is not cascaded; instead the switch provides full bandwidth between all the nodes. At the time of deployment, it cost approximately $430 to add one FX1400 to a node. In comparison, a new cluster node with an identical configuration but without a GPU cost approximately $4,000, not counting infrastructure such as free ports of the InfiniBand switch. No additional cooling infrastructure was required to support the graphics cards, and the cluster has proved to be as stable as systems without graphics cards. Additional maintenance tasks are limited to the occasional installation of updated device drivers for the GPUs.

The graphics cards in DQ are two generations old and current cards significantly surpass both computational performance and memory capacity, see Section 1.3. We use this hardware configuration because it is the only one available to us allowing scalability tests with so many nodes. Accordingly, we do not expect these GPUs to gain much performance, but rather help to provide insights into the behavior of large-scale heterogeneous systems that can be used in the planning of future clusters.

2.2 Solution procedure

It is well known that unstructured, fully adapted grids offer maximum flexibility in the discretization of PDEs and require an ’optimal’ number of degrees of freedom to achieve prescribed accuracy goals. The price for this is high though, since such grids lead to an almost ’stochastic’ numbering of the unknowns. The resulting linear algebra components, such as matrix-vector multiplication, perform very poorly on modern CPUs due to memory indirections and the implied cache thrashing; performance numbers less than 1% of the peak compute rates and less than 10% of the peak memory bandwidth on commodity architectures are common. In particular, performance gets worse as systems become larger. This holds true for the broad range of linear systems arising from Finite Element, Finite Volume or Finite Difference discretizations of PDEs (like the Poisson problem, cf. Section 3.1). On the other hand, generalized tensor-product grids (linewise numbering of the unknowns with fixed connectivity of the elements) yield banded matrices with a priori known structure, and allow for the design of much faster linear algebra components that explicitly take the matrix structure into account. Obviously, covering an arbitrary domain with one tensor-product grid is by far not flexible enough and inflates the amount of unknowns required to achieve prescribed accuracy goals. This is the first observation that lead to the development of the Feast package [2, 18, 35]. Feast covers the computational domain with an unstructured coarse mesh of patches called macros. Each macro is then refined independently in a generalized tensor-product fashion, and combined with r- and patchwise h-adaptivity and anisotropic refinement strategies, much of the flexibility of completely unstructured grids is recovered. In summary, Feast exploits locally structured parts of the discretization to achieve almost optimal (serial) performance with respect to the available bandwidth (cf. Section 1.3) by providing carefully tuned linear algebra components tailored to the discretization properties.

In parallel computations, the macros are distributed among the available nodes. The macros are coupled by special (inner) boundary conditions with implicit overlap: Only data on the macro edges needs to be exchanged and averaged to perform global operations. Adjacent macros on the same node communicate directly, neighboring macros on different nodes via MPI. In order to avoid the degradation of powerful multigrid smoothers in parallel and the poor scalability of pure domain decomposition techniques, Feast uses a global, data-parallel multigrid solver which executes the cheapest possible cycle and employs local multigrid solvers on each macro as smoothers. This approach is highly advantageous: The global solution procedure is as decoupled as possible, with the bulk of the work being performed locally. Synchronization between neighboring macros only takes place after smoothing, in the grid transfer operations and the coarse grid solver of the global multigrid iteration. This leads to very good scalability by design, since computation almost always dominates communication except for degenerate cases.

To solve the coarse grid problems, the local multigrid uses preconditioned conjugate gradient iterations, and the global solver an optimized sparse LU solver from the UMFPACK package [9]. Finally, the cascaded multigrid scheme is used as a preconditioner to a global Krylov subspace method, which is beneficial with respect to robustness of the scheme as a whole. Without it, we would either need more multigrid steps or a configuration with more expensive F-cycles in the parallel multigrid or both, resulting in more communication [2]. These three nested loops can be summarized as follows:

```
global BiCGStab, preconditioned by
  global multigrid (V-cycle, 1+1 steps), smoothed by
      local multigrid (F-cycle, 4+4 steps), smoothed by
          local Jacobi
```

Even though the FEAST project was started long before accelerators came into focus, the two main design features described above are crucial to making hardware acceleration possible. First, without explicitly taking the matrix structure into account, linear algebra components can not be implemented efficiently on GPUs (the accelerator in the scope of this paper). Memory indirections lead to more cache misses on GPUs (smaller caches) than on CPUs. Second, the decoupled solution scheme that performs the bulk of the work locally, without interruptions by communication, allows inclusion of the hardware co-processors on the node level. The local acceleration of the multigrid solvers is completely independent of the global solution process and all MPI communication. There is always enough local work to amortize the data transfer and configuration overhead of the co-processors. This minimally invasive integration is applicable for a wide class of accelerators. As we only have a GPU cluster available to evaluate the benefits with respect to weak scalability, power and performance, we have to restrict our tests to this hardware configuration.

Figure 2 illustrates this approach. Here, we only very briefly summarize the ideas and focus on the changes to the FEAST kernel while treating the GPU-based multigrid solver as a black box. For a more detailed description of the implementation and a discussion of various tradeoffs, we refer to previous work [12].

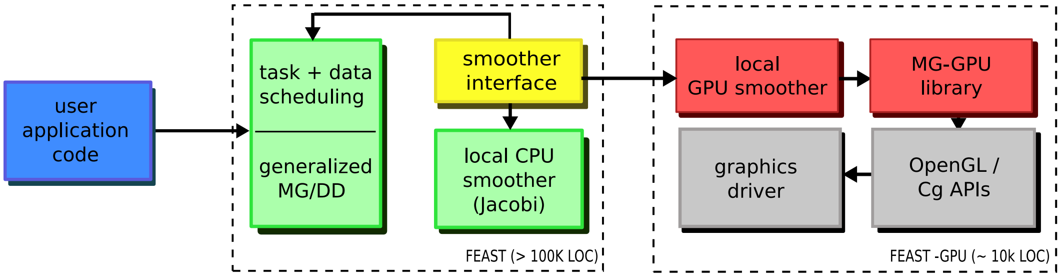

Fig. 2. Illustration of the hardware integration

The GPU local smoother implements the same interface as the existing FEAST smoothers; hence it can be selected and configured via a small change in a parameter file, in the same way as conventional smoothers. A lean interface layer accepts pointers to FEAST data in memory, converts it to single precision and passes it to the multigrid solver on the GPU. All implementation-specific details (OpenGL, shaders etc.) are completely hidden by encapsulating the GPU solver in a library, even from the main FEAST kernel. Consequently, backend implementations for other hardware can be added with little effort and can be developed independently of Feast.

The inner multigrid iteration described in the previous Section is executed either on the CPU(s) or the GPU on each node. If the solver executes on the CPU(s) then the GPU is completely idle. If the solver executes on the GPU, then one of the CPUs is also involved in the processing to some extent: It orchestrates the computation on the GPU, solves very small inner problems for which the transfer overhead is too large to justify an execution on the GPU, and is involved in the computation of local contributions to global matrix-vector and vector-vector operations, including the necessary MPI communication. As the inner multigrid is only a small part of the global preconditioner, mainly solving residual systems of the global multigrid, the restriction of the GPU to single precision has no effect on the final accuracy (cf. Section 3.1). The scheme can be interpreted as a mixed precision iterative refinement approach. For a more detailed analysis of mixed precision solvers of this type even for very ill-conditioned systems we refer to a previous publication [13].

2.4 Limitations of DQ's GPUs

Perhaps the most critical aspect in the coupling of the accelerators and the nodes is the data transfer between them, because the data typically has to travel through a bandwidth bottleneck(compare the PCIe interconnect and GPU bandwidth in Section 2.1 and see also Figure 1 on the right). The decoupled approach that performs as much (local) computation as numerically feasible before data has to be exchanged alleviates this problem: The overhead of data transfer to the GPU can only be amortized by a faster execution time if there is enough local work between communication phases of the accelerator with the host.

The GPUs in our DQ cluster are unfortunately one generation of technology older than the rest of the system, so we do not expect substantial speed-ups. Comparing the peak bandwidths alone, we expect a factor of $19.2/6.4 = 3.0$ for the accelerated parts of the solution procedure minus the time to transfer data via the PCIe bus and other related overhead. Despite being restricted to these old GPUs, we want to gain insight into and predict the behavior of more modern versions. The observed peak bandwidth roughly doubles with each generation of GPUs (Quadro FX1000 9.6 GB/s, Quadro FX1400 19.2 GB/s, Quadro FX4500 33.6 GB/s, Quadro FX5600 76.8 GB/s). Thus, we want to lower the bandwidth requirements of the computation on the old GPUs. Additionally, these GPUs have only very limited on-board memory and run into problems when storing the full matrix data for large problems (cf. Section 3.6); the data required for a multigrid solver on a large macro just barely fits into the video memory. On better GPUs with more local memory, we can store the matrix data for more than one macro, and we still have enough memory available to software-pipeline the code to reduce delays due to data transfer (see [12] for previous work on few, but more powerful GPUs).

We therefore implemented two versions of the matrix-vector multiplication on the GPU. The first one, labeled *var*, transfers all nine bands of the matrix and hence requires a lot of (PCIe and memory) bandwidth and storage. The second one, labeled *const*, assumes that the bands are almost constant and only stores a stencil and the 81 values deviating from it (values of the macro edges, vertices and the interior for each matrix band). The actual matrix entries are reconstructed during each matrix-vector multiplication, substantially reducing bandwidth and storage requirements while slightly increasing the computational overhead. This second implementation relies on an equidistant discretization and is no longer applicable for macros that are not parallelogram-shaped or use anisotropic refinement. We should point out that we use a similar bandwidth-friendly implementation on the CPU so no artificial advantage is given to the GPU. For rectangular grids, a simple Jacobi smoother suffices for the inner multigrid. More powerful smoothers are currently only available on the CPU. Hence, the results presented in this paper are prototypical, and we will be able to treat more realistic configurations on better GPUs once these smoothers are implemented. We do not expect qualitatively different results compared to the simple test cases in this paper, as these improvements do not affect the global solver.

3 Results

3.1 Test procedure

For the scalability tests in this paper we solve the Poisson problem $-\Delta\mathbf{u} = \mathbf{f}$ on a rectangular domain $\Omega \subseteq \mathbb{R}^2$ with Dirichlet boundary conditions. This is a fundamental model problem for all elliptic PDEs of second order, and an important building block in advanced applications such as fluid dynamics or structural mechanics. The discretization based on the refined macros uses conforming bilinear Finite Elements. We define an analytic test function $\mathbf{u}_0 = -(x^2+y^2)\cdot x$ on Ω and prescribe the right hand side as the analytical Laplacian of this function. Thus we know that $\mathbf{u}_0$ is the exact analytical solution for the continuous PDE, and we can measure the integral L_2 error of the computed solution against it. All solvers are configured to reduce the initial residuals by six orders of magnitude.

The computations are executed on one master node and up to 160 compute nodes of the cluster DQ (cf. Section 2.1), which is the maximum number of nodes available to us. In order to investigate the *weak* scalability of the co-processor enhanced cluster we keep the load per node constant and increase the number of nodes. The load per node is chosen to use as much of the available main memory as possible. In addition to scalability, we are interested in the total time to solution for various configurations of the CPU(s) and GPU, and in the difference between the two matrix-vector products (cf. Section 2.4).

For all test series we have confirmed that despite the limitation of the GPU to single precision calculations, the results are accurate both in terms of final residuals and measured L_2 errors.

3.2 Notation

For the different test configurations we use the following notation:

$$A\mathbf{g}P\mathbf{m}\_C\mathbf{n}\_L\mathbf{R}\_M \qquad \text{or} \qquad B\mathbf{c}P\mathbf{m}\_C\mathbf{n}\_L\mathbf{R}\_M$$

$A \in \{0, 1\}$ is the number of GPUs used per node,
$B \in \{0, 1, 2\}$ is the number of CPUs used per node,
$P \in \{0, \ldots, 16\}$ is the number of macros per node,
$C \in \{8, \ldots, 160\}$ is the number of compute nodes,
$R \in \{9, 10\}$ is the refinement level per macro, yielding $(2^R + 1)^2$ unknowns,
$M \in \{\text{const}, \text{var}\}$ is the type of matrix-vector multiplication.

Examples: 1g8m_16n_L10_const denotes a 16 node configuration where each GPU computes 8 macros with 1 Mi DOFs (degrees of freedom) each, using the stencil-based matrix-vector multiplication; 2c32m_64n_L9_var is a 64 node configuration where both CPUs in each node are used and each one computes 16 macros (32 per node) with 0.25 Mi DOFs each. A 1g_1c configuration is not considered because previous results show that it behaves very similarly to the GPU only (1g_0c) configuration [12]. The total number of unknowns can be derived from R, C and P. Except for one test configuration, we always use the fixed numbers of 2 Mi (L9) or 8 Mi (L10) unknowns per node. The largest overall problem size of the runs is therefore 1280 Mi DOFs.

3.3 Weak scalability and power considerations

Figure 3 compares the weak scalability of the DQ cluster using CPUs or GPUs. The test series increases the number of nodes (unknowns) from $C = 8$ (64 Mi) to $C = 160$ (1280 Mi). We compare the configurations where either only the GPU, only one CPU or both CPUs are active. All three test cases scale very

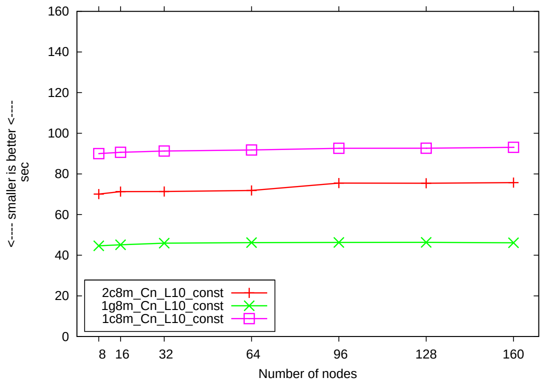

Fig. 3. Performance gains by the GPU and weak scalability (constant workload of 8 Mi DOFs per node).

well, and the rather outdated GPU outperforms even two (newer) CPUs by 66%. The reason for the advantage of the GPU becomes clearer in the next Section.

Under full load, the base node consumes approximately 315 W and the GPU consumes 75 W, so the addition of the GPUs increases the power requirements of the cluster by 20%, which is substantially smaller than the increase in performance. Newer generation GPUs generally not only improve raw performance but also performance per Watt, so adding GPUs to an existing cluster is clearly a worthwhile alternative to a more traditional cluster extension in this metric. The improvements in performance per dollar (and in performance gain per invested dollar) are also remarkable. Due to the limitations of the GPUs (cf. Section 2.4), we are not able to perform any meaningful tests for strong scalability as well. FEAST has been shown to scale well [2], and we have previously performed limited strong scalability tests on few, but more powerful GPUs [12]. Thus, we can assume that the strong scalability holds true on more GPU nodes as well. Under this assumption, we can estimate how an accelerated GPU cluster will scale in the metric 'energy to solution'. As we do not have exact numbers, we calculate an energy efficiency ratio for strong scalability, which we define as the ratio of modified and base system speed divided by the same ratio for power consumption (higher values are better). For the outdated GPUs in DQ, we achieve an efficiency ratio of 1.67 (doubling of speed, 20% increase in power consumption). The ratio for doubling the number of unaccelerated nodes is 1.0 (doubling execution speed, two times the power consumption) plus some epsilon as each node requires only half the amount of memory and hence slightly less power. We estimate 5 W/GB power consumption for the memory modules in DQ. The higher power consumption of newer GPUs is balanced by their speed: NVIDIA's Quadro FX 4500 GPUs consume 95 W each and execute the test problems of this paper (without the stencil-based matrix-vector multiplication) three times faster than the CPUs [12], achieving an efficiency ratio of roughly 3.0/1.3 = 2.3. Consequently, GPUs are also favorable in view of the costs related to operating a cluster, not only in terms of acquisition cost.

3.4 Impact of system bandwidth

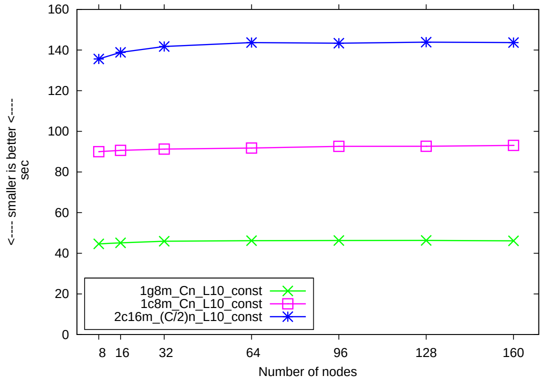

Fig. 4. Impact of system bandwidth (constant workload of 8 Mi and 16 Mi DOFs per node).

Figure 4 compares configurations with the same number of unknowns per processor whereas in Figure 3 the number of unknowns per node is constant. Thus, if speed of data access were the same, then the 1c8m_Cn and the 2c16m_C/2n should perform equally. However, the chipset in DQ's nodes shares the memory bus among the processors and the solution of the Poisson problem is bandwidth bound, such that the 2c16m_C/2n configurations perform far worse. The 2c16m_C/2n configuration performs slightly better for fewer nodes because domain boundary macros that require less arithmetic work are partitioned more evenly.

Similarly, the success of the GPU configurations has less to do with the parallelism in the GPU than with the superior bandwidth on the graphics card. The critical performance factor for this type of problem in a cluster is the overall system bandwidth, and the graphics cards add a lot of bandwidth to the system. Theoretically, the bandwidth increases by a factor of $19.2/6.4 = 3.0$. The impact of this advantage, however, is reduced by the low bandwidth connection of the graphics card to the main memory. Here the 'interconnect' between the distributed memories becomes the bottleneck. Therefore, the arrangement on chipsets with integrated GPUs, featuring shared memory for the CPU and GPU, is attractive for heterogeneous scientific computing, but these chipsets are currently low cost solutions with weak GPUs and narrow buses from the GPU to the memory. With specialized graphics memory chips (GDDR-GDDR3), their direct placement on the board and shorter paths, high signal quality and thus superior bandwidth is clearly easier to achieve in the proprietary layout of the graphics card than for the standardized memory modules used in PCs. While desirable, high performance integrated chipsets are not as easy to design as one might think at first. Both AMD and Intel pursue a closer coupling of co-processors to the main memory with their Torrenza and Geneseo technologies [1,17] (cf. Section 1.2), so substantial improvement can be expected in the near future.

3.5 Impact of macro size and type of matrix-vector product

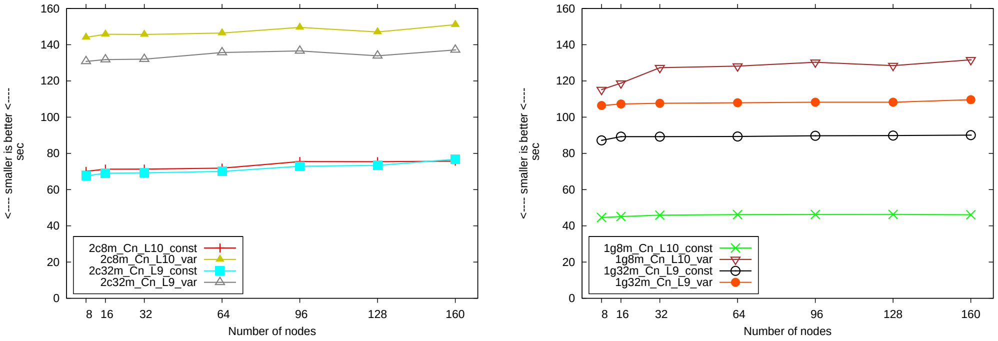

Fig. 5. Impact of macro size and type of matrix-vector product on the CPU (left) and the GPU (right), constant workload of 8 Mi DOFs per node.

Figure 5 (left) compares the scheduling of eight L10 macros against 32 L9 macros per node with both CPUs active, and the difference between the stencil-based and the full matrix-vector product. Figure 5 (right) makes the same comparisons for the GPU.

For the CPU and the bandwidth-friendly stencil-based matrix-vector product, we see that it makes almost no difference whether the problem is partitioned with macros of 1 Mi unknowns or with four times the amount of macros with 0.25 Mi unknowns each. The overhead of having four times as many calls into the inner multigrid solver is equaled by the reduced bandwidth requirement for the smaller macros, as a higher percentage of the operations is performed in cache. For the full matrix-vector product, the increased bandwidth requirements for the larger macros have a higher impact, and consequently, the computation takes 10% longer for the larger macros and the stencil-based computations are almost twice as fast due to the reduced bandwidth requirement. The effect is similar to a reduction of system bandwidth, see Section 3.4.

The numbers on the GPU tell a different story. The configuration with four times as many macros of one fourth the size is two times slower for the stencil-based matrix-vector product. Because the high number of macros implies a configuration overhead for the GPU, less parallel work is available in each step and the data is transferred via the PCIe bottleneck more frequently, but in smaller batches. In case of the full matrix-vector product more local work must be done on the small macros (L9_var) and the communication overhead is a smaller fraction of the overall execution time. Therefore, the distance between L9_const and L9_var is smaller than expected from pure bandwidth considerations and the same comparison on the CPU.

The graph of the L10_var configuration has several flaws. First, there is more parallel work in the L10 macros, so the L10_var should be faster than the L9_var configuration, while the two runs are switched in the graph. Second, in contrast to the other graphs, there is a higher variance for different numbers of nodes. The limited memory of the graphics cards is probably responsible for this strikingly different behavior. If data cannot be represented locally it must be moved back and forth through the narrow connection between the host and device memory (cf. Figure 1, right). Theoretically, there is (just) enough video memory for a L10_var macro, but we do not have sufficient control over its allocation and usage on the GPU to verify this. We attribute this problem to the memory capacity nonetheless, because graphics cards with more local memory perform very well in the L10_var configuration, as could be tested on up to 16 nodes [12].

Despite the anomalous timings discussed above, the graphs clearly demonstrate that the heterogeneous cluster enhancement with GPUs scales very well. Given more local memory on the graphics card these results should extend to the full matrix-vector product case, which is required for less regular grids and local operators other than the Laplacian.

3.6 Portability of the results to better hardware

A potential bottleneck of (massively) parallel multigrid solvers is the coarse grid solver on the unrefined macro mesh (see Section 2.2), which we perform on the master node while all compute nodes are idle. Since in our case it contributes less than 1% to the total runtime, we could add many more nodes before different approaches to solve the coarse grid problems, such as algebraic multigrid or even a parallel solve on multiple nodes, will become necessary.

As Feast's partitioning works with minimal overlap and data is only exchanged over macro edges residing on neighboring nodes, the net communication time is small. Separate timing measurements show that more time is spent in calls to `MPI_barrier()` for global synchronization than in the (asynchronous) movement of data between neighboring nodes. However, a much faster execution by the accelerator would increase the ratio between communication and computation. Similar to Amdahl's Law, further accelerating the computation results in quickly diminishing returns as soon as the computation is not the most time-consuming part. As we do not have enough newer graphics cards available to quantify the effect, we perform an 'inverse' experiment: Instead of increasing the communication to computation ratio of the cluster by newer and faster GPUs, we study the current GPUs with reduced network performance.

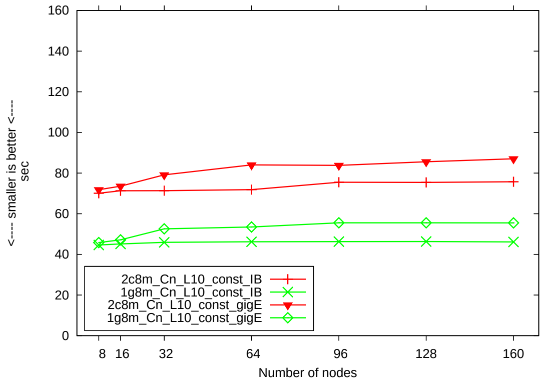

Fig. 6. Comparison of the performance using Infiniband and Ethernet interconnects.

Figure 6 shows that the type of interconnect has no significant impact on the weak scalability. Both the CPU and the GPU-accelerated runs suffer equally from the reduced network performance. Ultimately, the reduced computation time of ever faster accelerators must lead to the situation where the communication becomes the bottleneck, but we expect that we could continue to insert newer generation GPUs for several years and the current network would not have to be replaced. This is just another statement of the paradigm that data movement is expensive while computation is cheap.

The Ethernet results suffer from an additional penalty because the nodes are not fully connected; instead, some of the switches are cascaded. In addition, not all nodes are connected with 1 Gbps Ethernet, as a small number of them use the even narrower 100BaseT interconnects. Separate measurements with the MPI ping pong benchmark indicate more than a factor of ten in performance between the Infiniband and Ethernet subsystems. This helps to understand the faster execution for less than 32 nodes, but does not affect the discussion in the previous paragraph.

3.7 Future hardware

In this last Section, we discuss the anticipated impact of future hardware features on our approach. General limitations of the applicability of GPUs in scientific computing are discussed by Sheaffer et al. [28]. If we were designing a new cluster from the ground up, more than one GPU per node would certainly improve performance. We could easily assign an MPI job to each of them, and benefit from the fast shared memory communication between two processes on the same node. The benefits from high density GPU solutions such as NVIDIA's Tesla technology are similar. If at some point a direct, fast connection from board to board becomes available (SLI and Crossfire technology does not help here), we could only benefit from it if these GPUs were supporting double precision as well. We could then perform the correction iteration and shared memory communication in a much more efficient way, using the CPU(s) only for MPI communication. Similarly, RDMA support for GPUs would enable direct InfiniBand messages to and from video memory, bypassing the operating system. The most significant speed-up would result in removing the PCIe bottleneck. Both Intel and AMD have announced a more powerful coupling of processors, co-processors and main memory (see Section 3.4) and access to main memory through these designs is expected to be much faster and more flexible.

4 Conclusions and Future Work

We have demonstrated that a heterogeneous cluster with GPUs as co-processors scales favorably with the number of nodes and is also beneficial in metrics such as performance, power, performance/power, energy consumption and performance/dollar. We emphasize that this can be achieved without a fundamental refactoring of the initial homogeneous software package; in fact only 1% of the code has been modified to allow the interaction with the accelerator [12]. Restrictions of the GPU, such as data handling and precision, have absolutely no effect on the application code, and the accuracy of results is unaffected. Our tests are limited due to shortcomings of the outdated, available graphics cards, but from previous work we know that the general case can be handled equally well in this heterogeneous setup with newer graphics cards. In fact, the reduced memory requirements of an implementation tailored for orthogonal grids allows us to quantify the effects of system bandwidth, weak scalability and energy consumption even better. We believe that investigations like ours are important, as heterogeneous computing on a large scale is largely unexplored, at least in the Finite Element community.

In the future, we plan to use the results from this and previous work to accelerate real world applications with our heterogeneous approach. As the overall approach is independent of the type of accelerator, we will also investigate other co-processors of interest, such as the Cell.

Acknowledgements

We would like to thank our colleagues at Dortmund University and Los Alamos National Laboratory for help and support, in particular Brian Barrett, Christian Becker, Markus Geveler, Susanne Kilian, Jason Mastaler, John Patchett, Thomas Rohkämper, Galen Shipman and Hilmar Wobker. Thanks to NVIDIA and ATI for donating hardware that was used in developing the serial version of the GPU backend, and thanks to Mark Harris and Mike Houston for clarifying hardware details. We also thank the anonymous reviewers for their valuable and thoughtful suggestions on improving the paper.

References

[1] AMD, Inc., Torrenza technology, `http://enterprise.amd.com/us-en/AMD-Business/Technology-Home/Torrenza.aspx` (2006).

[2] C. Becker, Strategien und Methoden zur Ausnutzung der High-Performance-Computing-Ressourcen moderner Rechnerarchitekturen für Finite Element Simulationen und ihre Realisierung in FEAST (Finite Element Analysis & Solution Tools), Ph.D. thesis, Universität Dortmund, Fachbereich Mathematik (2007).

[3] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan, Brook for GPUs: Stream computing on graphics hardware, ACM Transactions on Graphics (TOG) 23 (3) (2004) 777–786.

[4] A. Buttari, J. Dongarra, J. Kurzak, Limitations of the PlayStation 3 for high performance cluster computing, Tech. rep., University of Tennessee Computer Science, CS-07-594 (2007).

[5] A. Buttari, P. Luszczek, J. Kurzak, J. Dongarra, G. Bosilca, SCOP3: A rough guide to scientific computing on the PlayStation 3, Tech. rep., Innovative Computing Laboratory, University of Tennessee Knoxville, UT-CS-07-595 (2007).

[6] ClearSpeed Technology, Inc., ClearSpeed Advance Accelerator Boards, `www.clearspeed.com/products/cs_advance/` (2006).

[7] Cray Inc., Cray XD1 supercomputer, `www.cray.com/products/xd1` (2006).

[8] G. Da Graça, D. Defour, Implementation of float-float operators on graphics hardware, in: 7th Conference on Real Numbers and Computers, RNC7, 2006.

[9] T. A. Davis, A column pre-ordering strategy for the unsymmetric-pattern multifrontal method, ACM Transactions on Mathematical Software 30 (2) (2004) 165–195.

[10] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stover, GPU cluster for high performance computing, in: SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, 2004.

[11] Genomic Sciences Center, RIKEN, The GRAPE series of processors, http://mdgrape.gsc.riken.jp/, http://www.metrix.co.jp/grapeseries.html, http://www.riken.jp/engn/r-world/info/release/press/2006/060619/index.html.

[12] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, H. Wobker, C. Becker, S. Turek, Using GPUs to improve multigrid solver performance on a cluster, accepted for publication in the International Journal of Computational Science and Engineering.

[13] D. Göddeke, R. Strzodka, S. Turek, Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations, International Journal of Parallel, Emergent and Distributed Systems 22 (4) (2007) 221–256.

[14] GPGPU, General-purpose computation using graphics hardware, http://www.gpgpu.org (2007).

[15] GraphStream, Inc., GraphStream scalable computing platform (SCP), http://www.graphstream.com/ (2006).

[16] M. Harris, Mapping computational concepts to GPUs, in: M. Pharr (ed.), GPUGems 2 : Programming Techniques for High-Performance Graphics and General-Purpose Computation, chap. 31, Addison-Wesley, 2005, pp. 493–508.

[17] Intel, Inc., Geneseo: PCI Express technology advancement, http://www.intel.com/technology/pciexpress/devnet/innovation.htm (2006).

[18] S. Kilian, ScaRC: Ein verallgemeinertes Gebietszerlegungs-/Mehrgitterkonzept auf Parallelrechnern, Ph.D. thesis, Universität Dortmund, Fachbereich Mathematik (2001).

[19] M. McCool, S. Du Toit, T. S. Popa, B. Chan, K. Moule, Shader algebra, ACM Transactions on Graphics (TOG) 23 (3) (2004) 787–795.

[20] Mercury Computer Systems, Inc., Cell BE accelerator boards, http://www.mc.com/microsites/cell/, http://www.mc.com/microsites/cell/ProductDetails.aspx?id=2590.

[21] H. Meuer, E. Strohmaier, J. J. Dongarra, H. D. Simon, Top500 supercomputer sites, http://www.top500.org/ (2007).

[22] F. Mueller, J. Weston, NSCU PlayStation 3 computing cluster, http://www.csc.ncsu.edu/news/news_item.php?id=464.

[23] National Center for Supercomputing Applications, Computer Science, University of Illinois, Scientific computing on the PlayStation 2, http://arrakis.ncsa.uiuc.edu/ps2.

[24] NVIDIA Corporation, NVIDIA CUDA compute unified device architecture programming guide, `http://developer.nvidia.com/cuda` (Jan. 2007).

[25] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, T. J. Purcell, A survey of general-purpose computation on graphics hardware, Computer Graphics Forum 26 (1) (2007) 80–113.

[26] V. Pande, Stanford University, Folding@Home on ATI GPUs, `http://folding.stanford.edu/FAQ-ATI.html` (2006).

[27] M. Peercy, M. Segal, D. Gerstmann, A performance-oriented data parallel virtual machine for GPUs, in: ACM SIGGRAPH 2006 Conference Abstracts and Applications, 2006.

[28] J. W. Sheaffer, D. P. Luebke, K. Skadron, A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors, in: T. Aila, M. Segal (eds.), Graphics Hardware 2007, 2007.

[29] Sony, Toshiba, IBM, Cell BE processor and blade systems, `http://www-03.ibm.com/technology/splash/qs20/`, `http://www.ibm.com/developerworks/power/cell`.

[30] Stanford University Graphics Lab, GPUbench – how much does your GPU bench?, `http://graphics.stanford.edu/projects/gpubench/results` (2006).

[31] R. Strzodka, M. Doggett, A. Kolb, Scientific computation for simulations on programmable graphics hardware, Simulation Modelling Practice and Theory, Special Issue: Programmable Graphics Hardware 13 (8) (2005) 667–680.

[32] D. Tarditi, S. Puri, J. Oglesby, Accelerator: Using data parallelism to program GPUs for general-purpose uses, in: Proceedings of the 12th international conference on architectural support for programming languages and operating systems 2006, 2006.

[33] A. Thall, Extended-precision floating-point numbers for GPU computation, in: SIGGRAPH '06: ACM SIGGRAPH 2006 research posters, 2006.

[34] Tokyo Institute of Technology, Global scientific information and computing center, `http://www.gsic.titech.ac.jp/`.

[35] S. Turek, C. Becker, S. Kilian, Hardware–oriented numerics and concepts for PDE software, Future Generation Computer Systems 22 (1-2) (2003) 217–238.